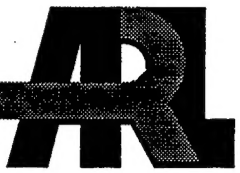
ARMY RESEARCH LABORATORY



Research Directions in Real-Time Systems

Michael J. Markowski

ARL-TR-1196 September 1996

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1996	Final, Aug-Nov 94

4. TITLE AND SUBTITLE

Research Directions in Real-Time Systems

5. FUNDING NUMBERS

622783.094

6. AUTHOR(S)

Michael J. Markowski

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-IS-TP
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-1196

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report summarizes a survey of published research in real-time systems. Material is presented that provides an overview of the topic, focusing on communications protocols and scheduling techniques. It is noted that real-time systems deserve special attention separate from other areas because of their unique properties. This has not been the case in computer science and operations research until quite recently, resulting in the absence of formal tools for design and analysis of real-time systems. The early work on applications as well as notable theoretical advances are summarized. Descriptions of more recent work in the field from the mid-1980s to the present are also provided.

14. SUBJECT TERMS

literature survey, real-time communications

15. NUMBER OF PAGES

21

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK.

1. Introduction

Computer systems are assigned ever more demanding jobs requiring networks of cooperating processors to complete tasks. In most cases, the goal is to finish as quickly as possible with no explicit deadline. It is with this implicit model that computer systems have traditionally been designed. When it is necessary for a system to be responsive to external events, the problem is typically mapped to some already solved non-time-constrained problem. Because of the relative success of this method, until recently, little research had been undertaken to determine the requirements of a real-time computer system. As a result, real-time system design is often not effectively undertaken to guarantee that the system can meet the application's demands.

A multidimensional problem, e.g., data with deadlines, precedence constraints, and resource requirements, in addition to priority levels, has its system requirements mapped into a single number, its overall priority in the system. The mapping necessarily loses important information about the data. The resulting system then has to be tuned by hand using both large simulations and the human experience from other system designs. It is common for unforeseen race conditions to exist that rarely occur, but wreak havoc when they do. Additionally, formal methods to design or analyze real-time systems are in their infancy. More research in the many subareas of real-time systems is needed so that formalisms and techniques can be developed.

The main differentiating factor between a real-time system and other types is that it responds to the changing state of its surrounding environment. Responding to real events means that data must be processed in a timely manner. Furthermore, no single processor can support the system functionality requirements, so processors must communicate to effectively transport tasks and data among themselves.

1.1 Real-Time Computer Systems

Real-time systems are the foundations of many applications: military command and control, automated manufacturing, process control, avionics, and numerous others. One can think of a real-time system as being made up of two major components: a *controlled system* and a *controlling system* [23]. The controlling system bases its decisions on data it receives from its subsystems, e.g., sensor banks, and because the decisions have physical impact, timely response is critical. The sensor/processing/actuator interconnection forms a feedback loop that sounds very much like one from traditional control theory. However, in general, real-time systems cannot be thought of as simply computerized control theory problems. Complex real-time systems will not have a simple Proportional Integral Differential (PID) control, but require control decisions based on complex rules and heuristics as are found in many modern process control plants. The computer closes the feedback control loop. As a result, when distributed nodes must coordinate their actions to make a decision, or when a database is updated or other similar tasks, these actions are not handled by feedback control theory which usually assumes a linear system. Because of the complexity of today's real-time systems, guaranteed optimal solutions are unavailable. Therefore, optimal eventual satisfaction of a request is less important than suboptimal, timely response.

Because deadlines cannot always be met, and because faults occur occasionally, the system must be able to make compromises to avoid a catastrophic situation. This is clear considering examples of controlling systems for airplanes or nuclear plants. But even with this major factor in common, types of real-time systems vary widely. Tasks can be periodic or aperiodic. Some tasks in a real-time system will have no deadlines, but must be able to coexist with those that do. Furthermore, time is not the only constraint under which a task is executed. A task might need access to hardware subsystems such as I/O, or to data structures and even databases. If a task is broken into a set of subtasks, precedence constraints will likely exist. Another property of the physical impact of real-time system decisions is that some tasks are more critical than others. In those cases, redundancy may be needed, or at the least, some deadlines must be given higher priority than other tasks.

The design of real-time systems today is usually done in an ad hoc manner. Based on system requirements, tasks are given different priorities that incorporate deadline and criticality. This is a carryover from scheduling methods used in timesharing systems. The system is run or simulated, and then tuned until an

acceptable priority assignment is found. This is both error prone and time consuming. An ideal environment for the analysis and design of real-time systems is multi-faceted. Compilers are needed that support the concept of tasks and can calculate worst-case computation times of a task. Small real-time kernels are needed with, among other things, fast context switches, fast interrupt response, ability to lock code or data into memory, real-time clock, and special alarms and timeouts. Since nearly all systems rely on a communications network, protocols that support real-time requirements are a necessity. And, for the applications programmer, languages and compilers are needed. Finally, for the system designer, formal tools and techniques must be derived. The last underlies many of the other requirements and will be accompanied only with breakthroughs in the theory of modeling real-time systems.

Due to common misconceptions of which some are listed here, real-time systems have not attracted academic attention [22].

1. There is no science in real-time design.
2. Advances in supercomputer hardware will take care of real-time requirements.
3. Real-time computing is equivalent to fast computing.
4. Real-time programming is assembly coding, priority interrupt programming, and writing device drivers.
5. Real-time systems research is performance engineering.
6. The problems in real-time system design have all been solved in other areas of computer science or operations research.
7. It is not meaningful to talk about guaranteeing real-time performance because we cannot guarantee that the hardware will not fail and the software is bug-free or that the actual operating conditions will not violate the specified design limits.
8. Real-time systems function in a static environment.

While Stankovic addresses and refutes each point above, it is hoped that the work summarized in subsequent sections will be equally convincing in showing that each of the above points is unjustifiable and that research dedicated to real-time systems is needed.

1.2 Real Time LAN Architecture

This report is organized roughly following the well known OSI seven-layer protocol stack model. Rather than strictly following the definition for the data link layer, this layer is modeled by splitting it into the popular IEEE LAN sublayers: media access (MAC) and logical link control (LLC). All others follow the definitions of the OSI model.

Until real-time systems were given special consideration, real-time properties have often been built in to a system at the application layer, which would have knowledge of the inner workings of lower layers. This is exactly what layered design was developed to avoid. It also explains the difficulty and high cost associated with the design of a real-time system—every one is unique and designed from scratch. With recent design principles applied to real-time protocols, protocols should ultimately exist that will simply offer standardized real-time services so that a real-time system design can be restricted to the application layer providing cheaper, easier, and simpler system development. To provide these services, at the very least each layer of the protocol stack must support the concept of deadlines. Since this support starts at the bottom, the MAC layer, this report starts there as well and works up through the higher layers.

Complex real-time systems will require communication incorporating new services and protocols. Existing protocols that offer little or no support for transactions with deadlines do not allow higher level, application-driven, real-time processing. To have a predictable, dynamic system that honors deadlines, at

least two additional services are needed that are analogous to traditional connection-oriented and connectionless services. Critical messages that are vital to proper functioning of the system must receive worst-case service guarantees. Less important messages with soft timing constraints should receive the system's best effort to transport it. Arvind et al. [2] introduce a LAN architecture, illustrated in Figure 1, which supports such services.

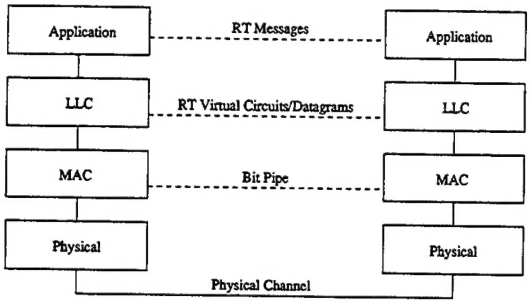


FIGURE 1.—*Typical real-time protocol stack.*

The Arvind model, RTLAN, is targeted for local control systems, thus internetworking is not considered. As a result, the RTLAN architecture layers jump directly from the LLC to the application layer. This report, however, also reviews the small amount of published work focusing on the network layer.

The RTLAN physical layer is identical to that of current LANs, since the change needed most is better system control rather than new supporting hardware. However, multiple physical channels may exist for redundancy and meeting performance or other functional requirements. Built on the physical layer, the MAC layer protocol will allow the LLC layer to offer real-time connection-oriented and connectionless services. Some of these MAC protocols are described in Section 2. The LLC in RTLAN has the difficult job of determining how, or if, to guarantee that tasks will meet their deadlines. It is at this layer that scheduling algorithms like those described in section 3.2 are implemented. And ultimately, it is at the application layer where time constraints are initiated. The requirements range from no time constraints, to soft constraints, up to hard real-time constraints. The levels of specialized services and protocols of RTLAN together strive to support the changing communication requirements of dynamic, distributed real-time systems.

2. MAC Layer

An integral part of a distributed real-time system is the communication network itself. Because of the time constrained nature of real-time data, protocols are required that offer worst-case guarantees for channel access for hard real-time systems, and at least statistically bounded blocking for soft real-time systems. Without these, higher level real-time algorithms—even if they provide theoretically optimal results—cannot be supported.

The most common type of network used in distributed computer systems is the multiaccess channel. Gong [11], provides an excellent overview of the most common (not real-time oriented) medium access protocols for the random access channel. The easiest MAC protocols to adapt to real-time use are controlled access protocols. Because access is explicitly under the control of the protocol, making worst-case guarantees is straightforward.

2.1 Token Ring

After random access LANs (Ethernet), token rings are the most popular type of LANs. However, the high-level, hand-tuned TDM approach used currently—round robin scheduling with each station getting a predetermined amount of bandwidth—creates fragile and inflexible systems. The tuning process itself,

where messages of differing levels of priorities and deadlines must be assigned queue positions, is expensive as well. Strosnider and Marchok [26] investigate a more flexible approach. It is necessary that messages can be directly assigned priorities by a scheduling algorithm. Also, enough levels of priority are needed to avoid degradation of scheduling and, ideally, resource preemption is necessary.

Because IEEE 802.5 (token ring) offers eight priority levels of which only four are available to the application, dynamic scheduling is difficult. Four levels do not offer enough resolution to make the fine distinctions between packets that real-time algorithms require. So, for synchronous tasks, a version of the rate-monotonic algorithm (discussed later in Section 3.2.3) is used. The original algorithm offers sluggish response time to asynchronous tasks. For his PhD thesis in 1988, Strosnider extended the rate-monotonic algorithm to one that guarantees that synchronous and alarm traffic will meet their deadlines, but also provides good response time for asynchronous traffic. The extended algorithm is called a deferrable server. Its performance is achieved by taking note that finishing a periodic task early in no way enhances system operation. Therefore, incoming aperiodic tasks are assigned higher priority than the periodic ones until a point is reached where the periodic tasks would begin missing their deadlines. At this point, any aperiodic tasks are given "background priority" until the start of the next period of the deferrable server when its available priority levels will be replenished. The results of a token ring simulation study are presented as validation of their performance improvement claim. The resulting ring can be mathematically analyzed, i.e., is predictable, and thus suitable for real-time systems.

2.2 CSMA-CD

By far the most common LAN is the randomly accessed channel, usually an Ethernet. Because of its widespread use, it is especially desirable to have protocols support real-time systems on this medium. Zhao and Ramamritham [20] consider four transmission policies for use in real-time systems distributed over a CSMA-CD network. The underlying protocol is *inference avoiding*, meaning that each transmission attempt is made without reference to past history of the channel or of that particular station, similar to Aloha and related protocols. However, the basic multiaccess scheme is augmented by using two clocks at each node. One gives real and the other virtual time. When the channel is busy, the virtual clock stops running and starts again when the channel is idle. When it does run, the virtual clock runs faster than the real clock. How much faster is a value to be determined based on expected traffic patterns. Depending on the variation used, earliest packet transmission (virtual) times are set to either arrival time, transmission time, latest time to send message so that it can reach its destination, or deadline. A message is sent only if its transmission time is less than or equal to the time on the virtual clock.

Laxity is defined, as in many real-time algorithms, as laxity = deadline − current time − processing time. Setting the virtual time to start transmission, as listed above, effectively implements these schemes in the same respective order: first- come, first-serve (FCFS), minimum length first, minimum laxity first, minimum deadline first. The results of simulating each protocol variation show that minimum laxity first and minimum deadline first perform better than the other two regarding delay and fraction of packets lost. It is interesting to note that in the static case where all information on tasks is available from the outset, Mok and Ward [17] showed that if some scheduling policy can successfully schedule a set of tasks, so can minimum laxity first and minimum deadline first. (A packet that is not successfully scheduled is blocked, or dropped.)

2.3 Splitting Techniques

Zhao, Stankovic, and Ramamritham [29] extend the work cited previously to a multiaccess protocol splitting protocol, i.e., one that makes transmission inferences based on collision history. Splitting protocols utilize either binary channel feedback, collision versus no collision, or ternary feedback, collision/success/idle. Most protocols of this type are recursive in nature, but require feedback only from the most recent slot. The protocol proposed by Zhao et al. uses the channel feedback history from the most recent two slots so that better predictions of upcoming channel usage can be made. Briefly summarizing the protocol, the actions taken based on history are:

- Collision: all transmissions aborted.
- Idle after collision: window split.
- Busy: waiting nodes continue waiting.
- Idle after transmission: window expanded.
- Continuing idle: window expanded, packet sent.

Simulations show that this algorithm closely approximates minimum laxity first and, importantly for real-time systems, performs well, even under overloaded conditions. Its performance is often better than the virtual time CSMA described previously and never worse. However, as with the virtual time protocol, laxity abnormalities still exist; when the laxity is relaxed, system performance may be degraded. This is not surprising, though, because with longer deadlines, more competing traffic is able to enter the system.

Arvind [1] simplifies the techniques from above and simulates splitting techniques for both hard and soft real-time systems. A simplification of the classical Gallager [8] ternary feedback splitting algorithm is used, except that servicing is not FCFS. Rather, it is based on a packet priority p, which is some unspecified function of its deadline and relative system priority that will have been assigned at some higher layer. The initial window encompasses the range of p. That is, following a collision, involved packets are ordered based on their priorities. Once ordered, that window is split so that highest priority packets are recursively transmitted. In this way, the order of successful packet transmissions is from highest to lowest.

In the hard real-time version, there are only a given number of virtual circuits. Nodes, at a higher protocol layer, contend for a virtual circuit assignment. Nodes with assigned virtual circuits can then contend for the channel in a typical random access scheme, except that collisions are resolved by assigning the integer virtual circuit number to p. The soft version is simpler in that it has no contention for virtual circuit assignments. The worst-case MAC time is generally greater than the hard real-time case because there will generally be many more nodes in the net than there are virtual circuits available.

An even simpler splitting technique appropriate for soft real-time systems is presented by Paterakis et al. in [19]. Rather than recursively splitting each window half, Paterakis instead only retains two cells of either collided or uncollided stations. Eventually all traffic will be either transmitted or dropped when some predetermined bound has expired.

2.4 Combining Real-Time with Non-Real-Time Traffic

Rather than find a specific transmission scheduling technique, some approaches use system requirements to determine a *class* of scheduling policies can meet the functional requirements of the system. A simple one-processor queuing model is studied by Delic and Papantoni-Kazakos [6] to find a class of nonpreemptive scheduling policies, i.e., policies in which packets being serviced cannot be suspended when higher priority packets arrive. There are two independent traffic streams modeled as two renewal processes, $\{X_i\}_{i\geq 1}$ and $\{Y_i\}_{i\geq 1}$. Y is a higher priority traffic stream with a strict upper bound $D_h^{\max}$ on its total delay from queue arrival to system exit. Arrivals are stored in a single infinite capacity queue.

The low priority traffic has a constant service time of c_l and the higher priority traffic a constant service time of c_h, where $c_h > c_l$. It is further assumed that since Y cannot be delayed more than $D_h^{\max}$, then $c_h \leq D_h^{\max}$. Also, for the system to be stable, for $Y \stackrel{\text{def}}{=} Y_{i+1} - Y_i$, the distribution of Y is such that $\mathrm{P}(Y < c_h) = 0$.

The class of scheduling policies to be determined is nonpreemptive and use only $D_h^{\max}$, c_h, c_l, and the oldest waiting times of high and low priority traffic, $W_{t_n}^h$ and $W_{t_n}^l$. The latter two variables take on negative values if no customers of that type are waiting.

Because of nonpreemption, minimum and maximum laxities are defined as

$$\mathrm{TH}_1 \stackrel{\text{def}}{=} D_h^{\max} - c_h$$
$$\mathrm{TH}_2 \stackrel{\text{def}}{=} \mathrm{TH}_1 - c_l$$

The general actions of any policy of the class are, at some t_n,

1. if $\mathrm{TH}_2 < W^h_{t_n} < \mathrm{TH}_1$, then a high priority customer is processed with probability 1.
2. if $0 \leq W^h_{t_n} \leq \mathrm{TH}_2$, a high priority customer is processed with probability p. If $W^l_{t_n} \geq 0$, a low priority customer is processed with probability $(1-p)$.
3. if $W^l_{t_n} \geq 0$ and $W^h_{t_n} < 0$, a low priority packet is processed with probability q.

Finding a stable system reduces to finding optimal values for p and q. Delic and Papantoni-Kazakos analyze a two dimensional, discrete time, discrete space Markov chain and discover that

$$q = 1, \qquad \forall W^h_{t_n} < 0, \quad \forall W^l_{t_n} \geq 0$$

Any value of p can be used and affects customer delays. (Delic and Papantoni-Kazakos further investigate system delays, but those results are not presented here.)

2.5 Soft Real-Time System with Performance Threshold

Since minimizing average response time, i.e., the time between a task being ready and the time it leaves the system, is an NP-complete problem, an optimal solution cannot be generated in real-time. (Section 3.2.1 gives a formal definition of the NP-complete nature of the problem.) If a soft real-time system takes the approach that under light load, a task can receive full service and under heavy load, only reduced service, then queueing analysis shows under what conditions stability can be maintained.

Liu et al. [13] analyze a simple model of a distributed, imprecise system with ν processors. In their model, "imprecise" means tasks can receive either full or reduced service. Results of occasional reduced service will be less than optimal, or imprecise. The model is an open ν-server Markov queue where tasks arrive according to a Poisson process with rate λ and join the common queue. Service rates are exponentially distributed, and tasks are served in a FCFS manner. A threshold H is chosen so that when less than H customers are in the system, system load is considered light. In this instance, the mean of the service distribution is $1/\mu$ corresponding to the average service time of the combined mandatory and optional subtasks. When system occupancy is H or more, this heavily loaded system offers a reduced-level mean service time that is a fraction γ of the light load service time. If all tasks are serviced at the light load level, the offered load is $\rho = \lambda/\nu\mu$. The system is stable as long as $\rho < 1/\gamma$, otherwise even at the reduced level of service, $\rho > 1$, and yields an unstable system.

While servicing all tasks at the reduced rate minimizes queue waiting time W, it maximizes system error. Studying the average fraction G of tasks serviced at the light load rate shows that $G = (U - \gamma\rho)/(1-\gamma)\rho$, where U is average processor utilization. With a good choice of H and when the offered load per processor is near unity, the system performs most effectively. In the corresponding precisely scheduled system, waiting time approaches infinity, whereas the two-level system keeps average waiting time small with a relatively small system error due to using imprecise results.

3. LLC Layer

While the MAC layer provides media access, in random access protocols it is usually in an unreliable manner. An LLC layer immediately above this would convert the unreliable service into reliable links between service access points. A typical LLC, e.g., the IEEE 802.2, provides three types of services: acknowledged connectionless, connectionless, and connection-oriented. These services, as will be described, can be thought of in real-time systems as being non-real-time, soft, or hard. Whether a system is hard or soft, providing reliable service now has the additional requirement of meeting deadlines.

3.1 End-to-End Guarantees within a LAN

It is difficult to guarantee meeting a task's deadline due to the dynamically changing performance of the underlying network. That is, in order for the system to function, process cooperation requires synchronization, which in turn relies on the communication network. Almost paradoxically, the component allowing the existence of a distributed system—the network—becomes itself a resource to be managed. The enqueuing of a message for transmission does not in and of itself mean that it will arrive at its destination by the deadline. Di Natale [7] addresses this problem with a technique offering end-to-end guarantees in a distributed, dynamic real-time system.

After off-line processing has determined communication time and precedence constraints of a system's tasks, local schedulers at each node are initiated. An *activation manager* interacts with the *processor scheduler* and the *network scheduler* so that tasks and intertask messages can be scheduled. In this scheme, there is no task migration between processors, only messages are passed. The processor scheduling is based on the bidding and focused addressing scheme described in section 3.2.3. The underlying network management that supports scheduling is divided into two layers. The MAC layer reserves a specified portion of the link's bandwidth for each node, and the LLC, when possible, generates a feasible network schedule for the messages at that node. To do so, network status is checked so that the message can be guaranteed timely delivery. Network scheduling has the added problem of determining that when an outgoing message is queued, the time awaiting service and transmission still allow the deadline to be met. Using these times in conjunction with the message's deadline, the network scheduler negotiates with the process scheduler to determine a message's transmission time. Since the network scheduler knows the outgoing message queue length at all times, the resultant message laxity is then used to determine the message's schedulability interval. Of the candidate intervals, the one maximizing the message's minimum laxity is chosen.

3.2 Scheduling

Tasks in a real-time system are constrained by several factors, chiefly time, resources, and precedences. The result is that finding an ideal order in which to execute the tasks becomes quite difficult. The goal of finding a feasible scheduling of tasks becomes even more challenging when the tasks are to be distributed over a network of processors. In addition to executing the tasks, interprocessor communication must be taken into account as well as the added delays of sending tasks from one processor to another. While some form of scheduling also takes place at higher layers, especially the transport layer, this section addresses only the problems encountered in scheduling tasks within the LLC layer.

Scheduling problems in general have been studied mainly from three directions: integer programming, heuristics, or queuing theory analysis. Integer programming offers potentially optimal solutions, but the problem complexity grows as an exponential function of the problem order. These methods are therefore almost never applicable to scheduling within dynamic real-time systems. Heuristic methods are the more promising approach in a real-time framework because though they offer suboptimal solutions, the solutions are generated quickly and are thus able to meet strict functional timing requirements. When a specific scheduling policy is not needed during the design phase, it is often useful to find a class of scheduling policies that meet system requirements. Here, queuing theory is a more fruitful approach than direct deterministic formulation. During implementation of a system, not surprisingly, heuristic methods are usually the most suitable to meet real-time constraints.

Scheduling is tailored for use in either hard or soft real-time systems. With a soft workload, results are wanted as quickly as possible, but there are fewer, if any, deadlines. An example of this might be a video application where degradation of a frame or even total loss of a frame on occasion is not detrimental. As in many other areas of computer science, scheduling algorithms can be classified as either *static* or *dynamic*. Static scheduling, or *off-line* scheduling, such as integer programming and other exhaustive techniques, is advantageous when applicable because the computationally intensive job of finding a schedule can be done ahead of time. Once the system is up and running, there is a low run time cost for the scheduling algorithm. However, the inflexibility of a static schedule is often its downfall. Due to failures, upgrades, refits, etc., a static schedule must be regenerated, and this is often not acceptable. While dynamic scheduling has higher

run time cost, it is more flexible and can adapt to a changing system.

Before detailing scheduling theory work, some elaboration on the nature of a task may be helpful. In most, though not all, scheduling problems, a task is considered an atomic entity. It is executed by a single processor. In addition to resource constraints, a task has associated timing constraints: arrival time, earliest time to begin execution, worst-case computation time, relative importance, and deadline. Tasks that are periodic generally are considered to have deadlines which are the times of their next invocations, while deadlines for aperiodic tasks are usually the times at which they must complete execution. Precedence constraints force ordered task execution. If task $T_i \prec T_j$, T_i must complete execution before T_j begins. If a task must run to completion once begun, it is nonpreemptable. Otherwise, it is preemptable. It is easier to generate a feasible schedule to meet deadlines when preemption is allowed. This is because small "holes" left in a partial schedule can be filled by portions of as yet unscheduled tasks, even though an entire task cannot fit. This makes meeting deadlines much easier. However, preemptable schedules are not always an accurate model of reality; a distributed database write, for instance, should not be preemptable. A consequence of this is that more research has been devoted to the preemptive case. And finally, tasks can be given a class designation, or priority, based on their relative importance in the system. Applications may have their own additional, unique constraints, but these listed are nearly universal in real-time systems.

3.2.1 NP-Complete Problem

Realizing that a problem appears to be difficult does not mean an optimal solution does not exist, but only that we are perhaps not clever enough to find it. NP-complete problems, formally defined shortly, are considered to be among the most difficult. This class has the property that if any *one* problem is shown to be tractable (solvable in polynomial time), then *every* problem in the class is tractable. However, though no proof has been found, NP-complete problems are considered intractable.

Formally, the complexity class NP is that class of decision problem languages that can be verified by a polynomial time algorithm. That is, for the polynomial time algorithm A and constant c, NP is defined as:

$$L = \{x \in \{0,1\}^*: \exists \text{ a verifying binary string, } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\}.$$

From this definition of NP, we can further define the most difficult set of problems in NP, those that are NP-complete. We say a language $L \subseteq \{0,1\}^*$ is NP-complete if

1. $L \in$ NP.
2. $L' \propto L$ (in polynomial time) $\forall L' \in$ NP.

Garey and Johnson [9] prove that scheduling under resource constraints is an NP-complete problem and thus computationally intractable. Their augmented multiprocessor model is composed of processors, resources, and tasks. The set of resources, $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$, is finite, and for each resource R_j, there exists a bound B_j of how much of the resource is available. Finally, there is the finite set of tasks, $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$, where each T_i is executed by a single processor.

Tasks are executed by a processor under three constraints. First, each task T_i has associated it with some value τ_i representing the amount of processing time the task requires. During execution, no other task can be served. Second, execution is based on a partial ordering of $\mathcal{T}$: if $T_i \prec T_j$ then task T_i must complete before task T_j can begin service. Finally, for all tasks and resources, $R_j(T_i) \leq B_j$. That is, each task has some nonnegative requirement of each resource. Furthermore, no subset of tasks simultaneously in service can require more than the total amount of a given resource.

The problem, based on these inputs, is to generate a schedule satisfying the constraints. We can further impose an overall system deadline D. An algorithm, f, generates a *valid schedule* if:

1. Each task completes by the deadline D, i.e., $f(T_i) + \tau_i \leq D$.
2. If $T_i \prec T_j$, then $f(T_i) + \tau_i \leq f(T_j)$.
3. A task T_i is served between $f(T_i)$ and $f(T_i) + \tau_i$.
4. For all tasks under service, no resource is requested exceeding its bound, B_j.

A consequence of the reducibility relation $\propto$ is that one need only show that a proven NP-complete problem reduces to the new problem. Using a two-processor model, only a single resource, each $\tau_i = 1$, and $\prec$ a forest, Garey and Johnson show that the well known Vertex Cover problem [4] reduces to multiprocessor scheduling. Though the proof is not included in this summary, the reduction is constructed as follows. Given an input for the problem, one constructs an m node graph G, each node representing a distinct task, with an edge connecting T_i and T_j if and only if $R_k(T_i) + R_k(T_j) \leq B_k$ for all k, $1 \leq k \leq r$. Tasks T_i and T_j can only be executed simultaneously on different processors only if an edge connects them. A valid schedule exists for this input if and only if $D \geq m - |E|$. Given G and positive integer k, there exists a vertex cover of size k for G. Thus multiprocessor scheduling is NP-complete.

Refer to Garey and Johnson [9] for the proof. While it may not be unexpected that the problem is NP-complete with an arbitrary number of resources, it is more surprising to note that the problem is NP-complete even with only a single resource.

Błażewicz, Drabowski, and Węglarz [3] extend these results. Their work shows that in the general, nonpreemptive case, even when tasks are subdivided and served by more than one processor, the scheduling problem is NP-complete. However, if the number of tasks is known to be between 1 and k, an integer linear programming formulation with a fixed number of variables is obtained. The complexity is this case is $O(n)$. Also, for preemptable tasks with arbitrary computation times, an $O(n)$ algorithm exists if the tasks require either 1 or some fixed k processors.

3.2.2 Reward/Penalty Coadaptation

An interesting solution to generating a feasible schedule is to use a coadaptive behavior heuristic. This is attractive because the system adapts to its own use of the net and so only requires some amount of time to reach stability. Therefore, no complex algorithmic techniques need to be built into the system. Its communication routes gradually evolve in such a way that a feasible, though suboptimal, schedule is generated. The idea put forth by Glockner and Pasquale [10] is that after a node joins the system and needs to send tasks or messages to another processor, it does so based on a set of probabilities. If node i successfully sends a task to node j and system performance improves, the probability of using node j again at some future time is increased. If system behavior is degraded, the probability is reduced. Determination of whether system performance has improved or not, however, is difficult to do in a distributed manner.

Simple equations are used for calculating probabilities. In an n node system, to reward a node for sending to some node i, the new probabilities become

$$\begin{aligned} p_i(t+1) &= p_i(t) + \alpha\big(1 - p_i(t)\big) \\ p_{j \neq i}(t+1) &= (1-\alpha)\, p_j(t). \end{aligned}$$

For a penalty, the probabilities become

$$\begin{aligned} p_i(t+1) &= p_i(t) + \beta\big(p_i(t)\big) \\ p_{j \neq i}(t+1) &= (1-\beta)\, p_j(t) + \frac{\beta}{n-1} \end{aligned}$$

The last term of the last penalty equation moves the automaton more quickly towards the equiprobable state.

Since system use of the communications network is used to generate its overall actions, the rate at which events occur and the sizes of rewards and penalties, i.e., the values of α and β, have a large impact on system behavior.

3.2.3 Scheduling in Hard Real-Time Systems

The major feature distinguishing a hard real-time system from other types is that, as its name indicates, deadlines are hard. That is, if the deadline is not met, the system cannot meet its functional requirements, possibly resulting in a catastrophe. Consider an assembly-line robot blocking the way of the next automobile on the line. It has only so much time to move to avoid a collision. If the robot cannot be moved in time, it is imperative that the system recognize this and schedule an emergency task to shut down the assembly line immediately. Schedules in a hard real-time system must guarantee that when a task is scheduled, it will be processed by its deadline. If that is not possible, then some acceptable degradation in service or fault tolerance is offered.

Rate-Monotonic Scheduling. For periodically executed tasks on a single-processor system, efficient algorithms exist that were developed early in scheduling research. In 1973 Liu and Wayland [12] analyzed a hard real-time system assuming:

1. Tasks with hard deadlines are periodic with constant period.
2. A task must complete before another request for it occurs.
3. Tasks are independent.
4. Computation time for a task is a constant.
5. Nonperiodic tasks do not have hard deadlines but can displace periodic tasks.

A deadline in this scheme (associated only with a periodic task) refers to the next instant that a task will be requested again. Scheduling algorithms are preemptive and priority-driven. In this system, a task experiences the longest service delay whenever a task is simultaneously requested with all other tasks of higher priority. As a result, it can be proven that a good heuristic is to assign higher priorities to tasks with higher request rates. This is called the *rate-monotonic priority assignment* and is frequently referred to in the literature. Furthermore, it can be shown that if a task set has a feasible priority assignment, then the rate-monotonic priority assignment is feasible.

It is also interesting to observe processor utilization rate. Liu and Wayland show that if the ratio between two request periods is less than 2, the least upper bound of processor utilization is $U = m(2^{1/m} - 1)$. In the specific case of two tasks, the maximum processor utilization possible is $\simeq 0.83$, while for large task sets, it is on the order of 0.70.

For more flexible scheduling, a dynamic, deadline-driven method is considered. The task whose deadline is nearest will be executed first. As long as the sum of task processor utilization is less than 1, the deadline-driven scheduling algorithm is feasible. If a feasible schedule cannot be generated, a processor will have no idle time before overflow. Using a combination of the rate-monotonic and deadline-driven scheduling algorithms allows the system to statically, i.e., rate-monotonically, schedule a subset of tasks while the rest are scheduled in the deadline-driven manner. While it is proven that this does not allow 100% processor utilization, it is much better than a purely rate-monotonic schedule and is the basis for many recent scheduling algorithms. However, it is rarely the case that tasks with deadlines are periodic and that task execution times are constant. Subsequent work has attempted to take this into account.

Preemptive Scheduling with Precedence Constraints. Several efficient algorithms have been found that generate optimal schedules in special cases. Muntz and Coffman [18] considered how to schedule tasks whose precedence structure is represented as an acyclic, directed graph. To generate the shortest length preemptive schedule, a reverse precedence tree is constructed weighting each node (task) with its worst-case computation time. The leaf nodes then are tasks that have no predecessors, while the root is the last task to start prior to the deadline. Processor time is assigned to leaf nodes highest in the reverse tree. As time

goes on and tasks become partially served, if their remaining service time moves them lower in the tree than other leaves, then those other leaves will be assigned processor time. The closer an unfinished task is to the leaves, the more processor time it will be assigned. This is an $O(n^2)$ algorithm and finds a minimal length preemptive schedule.

Preemptive Scheduling with Time and Resource Constraints. Even with a single resource, it is computationally intensive to generate schedules and especially to do so dynamically. The more complicated problem of scheduling n preemptable tasks in a system with r resources is considered by Zhao, Ramamritham, and Stankovic in [28]. Resources are usable in either shared or exclusive modes. Using combinations of simple heuristics, suboptimal schedules are generated in $O(rn^2)$ time, which is significantly lower than optimal algorithms.

In this multiprocessor model, a schedule for a set of preemptable tasks is viewed as a sequence of time slices. During any slice, several tasks can be running in parallel with each task assigned to one or more system resources. A *full schedule* is one in which every task is scheduled to run for at least as long as its required processing time (longer in the case of preemptions), and a *full feasible schedule* is a full schedule that completes before the deadline. The problem then is finding a set of slices that correspond to a full feasible schedule.

In a dynamic distributed system, tasks unpredictably arrive at nodes and require scheduling. The updated schedule, however, must still guarantee the completion of tasks previously scheduled. This algorithm addresses the problem of scheduling arriving tasks on a single processor assuming that an algorithm exists to effectively move tasks between processors.

The search space is a tree whose root is the empty schedule. Each additional descendant adds to its parent's partial schedule so that leaf nodes contain full schedules. With n tasks and a active resources (a resource with processing power usable by at most one task at a time) can have up to $\binom{a}{b}$ children. To make such a large search space computationally tractable, heuristics are used so that even in the worst-case, scheduling can be directed from the root to a full feasible leaf node schedule. Using the heuristics, a node at a level is described as *strongly feasible* or not. It turns out that if a node is not strongly feasible, no path from its children will lead to a full feasible schedule. It is possible, though, to reach a level with no strongly feasible nodes even though the tasks are schedulable. In this case, limited backtracking is used.

The heuristics base scheduling on arrival time, deadline, laxity, computation time, or other basic properties of a task. Depending on the application, one group of heuristics may make more sense than other. Simulations show, however, that heuristics utilizing minimum deadline first and minimum laxity first perform best.

Branch and Bound Task Allocation. For the Ballistic Missile Defense Advanced Technology Center and TRW, Ma, Lee, and Tsuchiya [15] developed a task allocation algorithm for a distributed system striving for minimal interprocessor communication cost, processor load balancing, and meeting real-time application requirements. Solutions by integer programming methods are limited by the fact that for scheduling, time and memory requirements grow as exponential functions of the problem order. Therefore heuristic methods are appropriate when an optimal solution is not computable within timing constraints. The method proposed is based on the branch and bound method, a zero–one programming technique. To verify its correctness, the algorithm was tested on an air defense system broken down into 23 real time tasks on 3 processors.

Detailed task preprocessing is necessary and requires that task interaction, amount of data communicated, computation time required, etc., must be known beforehand. There are several other required data structures: a *task preference matrix* indicating tasks that must be run on specific processor, a *task exclusive matrix* defining mutually exclusive tasks that cannot be run on the same processor, and *task redundancy* permitting multiple copies of a task for system reliability. Network preprocessing is also needed; thus, topology and link costs must be made available prior to system startup. The cost of using a link is based on the amount of data to transmit and the distance the data must travel. There is also a cost associated with processing a given task on a given processor.

With the above data, branch and bound method is implemented by viewing the allocation problem as a search tree. When m tasks are allocated among n processors, an m-level tree is generated with each node having n children, each representing a possible processor allocation. The search tree then has n^m nodes. However, many branches can be pruned using elimination rules. If a node at level k is eliminated, then n^{m-k} possible allocations are pruned. Pruning in this algorithm is necessarily dependent on application-specific constraints, so a lowest or even average upper bound is not obtainable. By checking partial costs during the search and comparing to some upper bound, expensive subtrees can be eliminated. Subtree elimination also occurs based on the preference and exclusive matrix values.

For the air defense problem, the algorithm finds an optimum solution in 10^4 iterations with an upper bound of $O(10^{10})$. Because this technique is heavily application oriented, it is not a generally useful one.

Nonpreemptive Scheduling with Bidding and Focused Addressing. Early research in scheduling focused on single and multiprocessor systems. With loosely coupled systems becoming the norm, i.e., LANs, the focus of research widened to include distributed systems. Stankovic, Ramamritham, and Cheng [25] introduced an effective algorithm for this environment combining two heuristics, *bidding* and *focused addressing*. Each node in the network runs a local scheduler which, when necessary, interacts with others to schedule tasks in a distributed manner. Each node is modeled as having a set of periodic tasks for which it is responsible. These tasks are scheduled and thus *guaranteed*. Arriving without notice from the network are aperiodic tasks that are scheduled when possible. Whenever a task is scheduled, it is guaranteed. That is, the processor has determined it has enough excess computational power to complete the task by its deadline. If a task cannot be guaranteed, other schedulers are contacted to see if one of them can guarantee the task.

The algorithm is made up of four tasks that all share data such as the periodic task table, system task table, list of requests for bids, and List of bids sent out. Aside from an initialization task, there is a dispatcher task, a bidder task, and a local scheduler task. The dispatcher is invoked each time a task completes. Of the scheduled tasks, periodic or aperiodic, the task with the earliest deadline is executed next. This does not provide an optimal, network-wide schedule, but is fast and reduces overhead. Using the periodic and system task tables, the local scheduler decides whether or not a newly arriving task can be scheduled.

When a task cannot be scheduled locally, the heuristics are employed. By keeping track of information on processor time surpluses, likely candidates to take on more tasks can be predicted. In a distributed environment with a large delay relative to processing speed, the information is always out of date so the focusing algorithm uses the results only as an estimate. If a candidate is found, the task is sent to it. If one is not found (and actually if one is found, too), the bidding scheme is begun. When a task cannot be scheduled locally, a request for bids is sent out. The task is sent to one of the processors which responds. The bid request and any responses are also used by other nodes to update their system tables. Simulations show that the combination of the two heuristics works quite well.

3.2.4 Scheduling in Soft Real-Time Systems

Not all tasks in a general real-time system have hard deadlines. When a deadline is soft, or desirable but not mandatory, that means a precise output is not required. Imprecise calculations can be useful as a means of implementing graceful degradation of system behavior. It is common, for instance, for a system to contain *sieve functions*, functions whose sole purpose is to produce an output at least as precise as its input or some older output value. For example, while processing a return radar signal, the step in each iteration that calculates a new estimate of the signal-to-noise ratio can be skipped and the previous value used [13]. Similarly, in process control or while tracking an object, some less precise result is better than none.

Soft scheduling can also be implemented when *monotone functions* are present. This is a function whose quality of intermediate results does not diminish as execution time increases. It is therefore possible to simply terminate task execution before completion. Whatever result has been calculated to that point is used as the output.

It is also possible to simply have multiple versions of tasks. A larger, more complex version offers a precise result, while a smaller, simpler version offers only an estimate, but in a short span of time.

Depending on the application, tasks with soft deadlines are either error cumulative or error noncumulative. Noncumulative are easiest to handle because their optional subtasks are always optional. Error cumulative tasks, like the object tracking example, on the other hand require special treatment. After some number of consecutive imprecise results, a precise result is required, and the optional subtask temporarily becomes a mandatory one.

Minimizing Imprecision. Liu et al. [14], [13] propose several ways of scheduling tasks of the above type where the standard model of a task is extended. Instead of being considered an atomic entity, it is now made up of two subtasks: a mandatory subtask and an optional one. This is analogous to the multiple task version just described. The pared down version can be thought of as the mandatory task, and the larger and more precise version as the optional task. From this point of view, it can be seen that general soft real-time systems are supersets of traditional soft real-time systems where all tasks are optional and hard real-time systems where all tasks are mandatory.

The soft scheduler first takes care of the mandatory subtasks then uses whatever time is left to schedule the optional subtasks. Any unscheduled optional subtasks contribute to system error. The scheduling of optional subtasks then can be done with an eye towards minimizing various types of error: maximum error, average error, discarded optional tasks, number of tardy tasks, average response time, and so on. Furthermore, when an imprecise result is fed into a successor task, that task in the best case can only generate an output with the same amount of error. If its result is made still more imprecise, one can see it is easy for a system's behavior to quickly degenerate. Error must be accounted for not only at the task level, but at the system level as well. As one would expect, the problem is easier when all optional tasks have equal weight.

In imprecise systems, the definition of a feasible schedule can be divided into two categories. If a schedule can be generated that provides processor time for all mandatory and all optional tasks, the schedule is *precise*. If all mandatory tasks are scheduled but optional tasks are only partially so, the schedule is said to be *imprecise*. The general framework of Liu et al.'s algorithm, named Algorithm F, is as follows:

1. Try to schedule all tasks treating them as optional. If the schedule is precise, stop. Otherwise, continue with step 2.
2. Try to schedule all mandatory tasks precisely. If this is not possible, stop; there is no feasible schedule. Otherwise, continue with step 3.
3. Transform the schedule from step 2 into one that allows as many optional tasks as possible processing time and minimizes total error.

As an aside, if tasks are independent and parallelizable, off-line scheduling can be calculated using integer programming. If it is assumed that multiprocessing overhead is a linear function of the degree of concurrency, formulating the system constraints is made easier. Integer programming is computationally expensive, however, and the current best algorithm is a polynomial time one by Karmarkar with a complexity of $O\big((\alpha+\beta)\beta^2+(\alpha+\beta)^{1.5}\beta\big)$, where α is the number of inequalities and β is the number of variables.

Imprecise Rate-Monotonic Scheduling. To implement rate-monotonic scheduling in a soft real-time system with tasks modeled as described previously, some modification to the algorithm is required. These modifications are put forth by Shih, Liu, and Liu in [21]. As in Liu and Layland's original algorithm, the modified version is also preemptive and priority-driven. Now, however, requests are divided into two groups: current requests and old requests. They are then prioritized using the following three rules:

1. Current requests have lower priority than old requests.
2. Priorities of each group are assigned in the original rate-monotonic way.
3. All old requests are scheduled in a first-in, first-out way.

The deadline of a task can be expressed as

$$d_{ij} = b_i + jp_i + \delta_i$$

where b_i is the ready time of the first request of the task, j is the jth request, p_i is the period of the task, and δ_i is the deadline deferral of the task. If γ is the ratio between the longest and shortest periods of all tasks, then the modified rate-monotonic scheduler is optimal when the deadline of every job is deferred by $\max(1, \gamma - 1)$ periods or more.

4. Network Layer

Very little published work addresses the problems associated with the network layer of a real-time system. Presumably this is because currently it is almost unheard of to internetwork real-time LANs. Most applications advertised as real-time are in fact soft real-time with loose timing constraints. These constraints, furthermore, are generally arrived at by taking into account the communications media over which the traffic must travel. The application is designed to fit the network rather than vice-versa. To some extent this is, of course, inevitable. An application cannot request higher performance than the network can deliver. However, protocols that take deadlines into account are make to offer better use of available bandwidth.

At the network layer this reduces to two major problems: finding routes and scheduling packet transmissions over them. There are many methods of generating routes that will not be elaborated on here. A distributed algorithm for mobile wireless networks by Corson and Ephremides [5] appears promising for use in real-time networks, because it generates all possible routes between a source and destination. This could easily be modified to generate routes while also collecting data such as link capacity, utilization, etc. Subbarao and Murali [27] present a literature survey of routing techniques applicable to mobile radio networks. They show that more work is needed for better route generation, especially in mobile nets with no backbone.

Once several routes are determined, however, it is necessary to calculate which ones can support the requested qualities of service. This is addressed by Zheng and Shin [30] for both preemptive and nonpreemptive scheduling. They present straightforward equations based on negotiated traffic rates, channel capacity, and requested delay. Based on resultant values, routes can be kept or discarded based on the applications needs.

5. Higher Layers

Currently, network management techniques do their best to offer real-time communications without real-time protocols. That is, traffic that exceeds predetermined thresholds for jitter, delay, and so on, are simply discarded. Given protocols through the network layer that support hard and soft real-time messages, it is expected that this will have little effect on network management. The difference will be that better service can be guaranteed (and charged for) that explicitly takes into account an application's needs rather than just hoping for the best as a side effect of the communications medium.

6. Other Areas

This report has emphasized real-time research areas in computer network protocols and their corresponding scheduling algorithms. However, research is ongoing in many other areas of real-time systems.

Languages and compilers are needed so that the applications programmer can indicate how processes communicate and, of special importance, under what constraints they must operate. The compilation must also divide a process into the smallest schedulable objects—tasks. Real-time languages must also deal with timing constraints by using a worst-case computation time for a task. Therefore, open ended loops,

unconditional branches, recursion, and the like cannot be allowed without careful control for tasks with hard deadlines.

Implementing timing constraints in a distributed system introduces another problem: clock synchronization. The only way processors' clocks can be synchronized is to read each other's clocks which is made difficult by communication delays, clock drift (typically on the order of 1 μsec/sec), and even clock failures. Mills [16], in his development of the Network Time Protocol, has shown that on an Ethernet or FDDI, it is possible to achieve reliable synchronization to within a few hundred microseconds. This is accomplished by driver and kernel modifications in conjunction with radio interfaces to allow synchronization with radio clocks.

Clearly, a real-time language is useful only if the computer's operating system supports such an environment. It is not enough for a kernel to simply be small and fast, though that is a good first step. A real-time kernel should be aware of a task's importance in the system and allocate resources when needed to groups of cooperating tasks. When a task cannot meet its deadline, the operating system would be able to offer an alternative rather than precipitating a system-wide failure.

The highest level of requirements are tools for design and analysis of real-time systems. Not only high level, conceptual ideas must be represented, but also descriptions of logical functioning and behavior, and even implementation level details. Real-time systems of today, however, are generally analyzed with large simulations. The main shortcoming with this approach is that only system bugs are discovered. Furthermore, it is difficult and time consuming to generate situations so that the simulator tests all states of the system. Small, subsequent modifications to the system usually have dramatic effects on its behavior not predictable from the initial results necessitating a new round of simulations. Formal verification tools that take into account both logical and timeliness properties of scheduling are thus necessary.

Some examples of current efforts in these and other areas are detailed in Stankovic and Ramamritham [24], [23].

INTENTIONALLY LEFT BLANK.

7. References

[1] K. Arvind. *Protocols for Distributed Real-Time Systems.* PhD thesis, University of Massachusetts at Amherst, 1991.

[2] K. Arvind, Krithi Ramamritham, and John A. Stankovic. A local area network architecture for communication in distributed real-time systems. *Journal of Real-Time Systems*, 3(2):115–147, 1991.

[3] Jacek Błażewicz, Mieczysław Drabowski, and Jan Węglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers*, 35(5):389–93, 1986.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, 1989.

[5] M. S. Corson and A. Ephremides. A distributed routing algorithm for mobile wireless networks. *Wireless Networks*, 1(1), January 1995.

[6] Hakan Delic and P. Papantoni-Kazakos. Class of scheduling policies for mixed data with renewal arrival processes. *IEEE Transactions on Automatic Control*, 38(3):455–459, March 1993.

[7] Marco Di Natale and John A. Stankovic. Dynamic end-to-end guarantees in distributed real time systems. In *Proceedings of the 1994 IEEE Real-Time Systems Symposium.* IEEE, 1994.

[8] R. G. Gallager. Conflict resolution in random access broadcast networks. *Proceedings of the AFOSR Workshop in Communications Theory Applications*, pages 74–76, 1978.

[9] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *Society for Industrial and Applied Mathematics Journal of Computing*, 4:397–411, 1975.

[10] Alexander Glockner and Joseph Pasquale. Coadaptive behavior in a simple distributed job scheduling system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):902–907, 1993.

[11] Yu Gong. *MAC Protocols in Communication Networks: Design and Performance Analysis.* PhD thesis, University of Delaware, 1992.

[12] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[13] Jane W. S. Liu, Kwei-Jay Lin, Wei-Kuan Shih, Albert Chuang-shi Yu, Jen-Yao Chung, and Wei Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[14] Jane W. S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, and Bettati Riccardo. Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94, January 1994.

[15] Perng-Yi Richard Ma, Edward Y. S. Lee, and Masahiro Tsuchiya. A task allocation model for distributed computing systems. *IEEE Transactions on Computers*, 31(1):41–47, 1982.

[16] David L. Mills. Precision synchronization of computer network clocks. *ACM SIGCOMM Computer Communication Review*, 24(2):28–43, April 1994.

[17] A. K. Mok and S. A. Ward. Multiprocessor scheduling in a real-time environment. *Proceedings of the Seventh Texas Conference on Computing Systems*, September 1978.

[18] R. R. Muntz and E. G. Coffman. Preemptive scheduling of real-time tasks on multiprocessor systems. *Journal of the ACM*, 17(2):324–328, August 1970.

[19] M. Paterakis, L. Georgiadis, and P. Papantoni-Kazakos. Full sensing window random - access algorithm for messages with strict delay constraints. *Algorithmica*, 4:318–328, 1989.

[20] Krithi Ramamritham and Wei Zhao. Virtual time CSMA protocols for hard real-time communication. *IEEE Transactions on Software Engineering*, 13(8):938–52, 1987.

[21] Wei-Kuan Shih, Jane W. S. Liu, and C. L. Liu. Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. *IEEE Transactions on Software Engineering*, 19(12):1171–1179, December 1993.

[22] John A. Stankovic. Real-time computing systems: The next generation. *Tutorial: Hard Real-Time Systems*, pages 14–37, 1988.

[23] John A. Stankovic and Krithi Ramamritham, editors. *Tutorial: Hard Real-Time Systems.* IEEE Computer Society Press, 1988.

[24] John A. Stankovic and Krithi Ramamritham, editors. *Advances in Real-Time Systems.* IEEE Computer Society Press, 1993.

[25] John A. Stankovic, Krithivasan Ramamritham, and Shengchang Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers*, 34(12):1130–43, 1985.

[26] Jay K. Strosnider and Thomas E. Marchok. Responsive, deterministic IEEE 802.5 token ring scheduling. *Journal of Real-Time Systems*, 1(2):133–58, 1989.

[27] Madhavi Subbarao and Ramaswamy Murali. A study of packet routing in mobile radio networks. Technical report, John Hopkins University, 1995.

[28] Wei Zhao, Krithi Ramamritham, and John A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, 36(8):949–60, 1987.

[29] Wei Zhao, John A. Stankovic, and Krithi Ramamritham. A window protocol for transmission of time-constrained messages. *IEEE Transactions on Computers*, 39(9):1186–1203, September 1990.

[30] Qin Zheng and Kang G. Shin. On the ability of establishing real-time channels in point-to-point packet-switched networks. *IEEE Transactions on Communications*, 42(2/3/4):1096–1105, 1994.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFO CTR ATTN DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL OP SD TA 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL OP SD TL 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LAB ATTN AMSRL OP SD TP 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
2	DIR USARL ATTN AMSRL OP AP L (305)

NO. OF
COPIES ORGANIZATION

1 DIRECTOR
US ARMY RESEARCH LAB
ATTN AMSRL IS TG J GOWENS
GIT 115 OKEEFE BLDG
ATLANTA GA 30332-0800

ABERDEEN PROVING GROUND

16 DIR, USARL
ATTN: AMSRL-IS-TP,
M. MARKSOWSKI (5 CP)
A. BRODEEN
F. S. BRUNDICK
H. CATON
S. CHAMBERLAIN
A. B. COOPER
A. DOWNS
D. GWYN
G. HARTWIG
R. KASTE
M. LOPEZ
C. SARAFIDIS

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-1196 (Markowski) Date of Report September 1996

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
U.S. ARMY RESEARCH LABORATORY
ATTN: AMSRL-IS-TP
ABERDEEN PROVING GROUND, MD 21005-5067